

Image Quality Sorter for digiKam

Gowtham Ashok

Short Description:

When we organize a large collection of pictures, we often come across both good and bad quality ones. These pictures can be computationally separated to provide better workflow. This project aims to integrate Image Quality Sorting capability into digiKam, so that we can focus on good quality pictures.

Name: Gowtham Ashok

Email Address: gwt93@gmail.com

Freenode IRC Nick: gwt93

IM Service and Username: xmpp:gwt93@gmail.com/Home, Username: gwt93

Location (City, Country and/or Time Zone):

Chennai, India, UTC + 5.30

Proposal Title:

To integrate sorting based on Image Quality in digiKam so that users can focus on high quality pictures.

Motivation for Proposal

When a large collection of pictures has to be organized, many often come across obviously bad quality pictures, along with good quality pictures. This impedes the workflow, and is frustrating, when dealing with 10000+ pictures. https://bugs.kde.org/show_bug.cgi?id=279544

This proposal seeks to alleviate this problem by separating pictures based on Image Quality. It is done so, by automatically judging the quality of an image by computationally considering various characteristics of the picture. This is analogous to the Spam Filter in GMail. It will also become easier for digiKam users to apply filters, such as Sharpen, since some of the specific picture defects can be found using the to-be implemented algorithm.

Project Goals:

- To conduct a Survey, asking developers and users to rate a sample set of

pictures.

- To write an Image Quality parser, which can be tested from command-line, as a standalone program.
- To patch Scan interface to call the Image Quality Parser.
- To patch Database schema and Database interface to register items through the parser.
- To patch GUI option to turn on/off parser at scan, add Quality Labels everywhere (special tags)

Importance of Image Quality Sorter in digiKam

Some pictures look good on the small digital camera display, but may have obvious flaws, which can be detected computationally. Image Quality Sorter tries to separate these pictures and assign tags to them for review later, allowing the user to focus on the determined good quality pictures first [15]. No automatic processing is performed on the original image. The result of this project can also be used to implement auto-enhance in digiKam like in GNOME's Shotwell [16]. In other KDE applications, the tags created by this project can be shared with rest of KDE through Nepomuk interface automatically. It then can be used in Dolphin to sort images according to Quality.

No open-source image editor has implemented separation of images based on Quality. If implemented, digiKam will be the first to do so.

Implementation Details

Algorithm to obtain Image Quality:

Image quality is a characteristic of an image which is affected by various factors, mainly Sharpness, Noise and Compression. To obtain Image Quality, there are Full-Reference and No-Reference methods. Since we need a reference image for the former, the proposal aims to use No-Reference method to evaluate Image Quality. Many research papers contain techniques to measure Image Quality.

My main reference paper is [5].

Blur:

Blur is caused mainly when the camera lens is not properly focussed on an object or due to atmospheric conditions.

This can be detected computationally by analyzing

- Fourier Spectrum
- Edges
- Fourier Statistics

Blur can be classified further into two types,

- **Isotropic blur** is the standard blur that occurs when the lens is out of focus in a camera.

To detect Isotropic blur, we calculate the ratio of the average edge density to the maximum edge strength.

- **Motion blur**, is caused by taking pictures of fast-moving objects.

To detect Motion blur, we compare the sharpened of the image with the original image, and if a high-degree of similarity is found, then motion blur occurs.

This proposal prioritizes detection of Isotropic blur.

The exact formulae, and detailed explanation are given in the research paper(Pages 9,10,11 [5]).

Methods specified in [6],[7] can be used to improve algorithm speed.

I have written a demo algorithm in openCV using Edges [1].

Noise:

Noise is mainly classified into

- Random
- Gaussian
- Salt-and-pepper

The amount of noise can be calculated by taking the average edge strength divided by the maximum possible edge value of an edge, provided we first determine that noise is present.

To detect if noise is present:

1. Take each pixel, compare its intensity value. If it is very different from its neighbour pixels, then that pixel contains noise. A threshold is set, above which noise is said to be present in the image.
2. Construct a histogram based on the shading values of the pixels, where different types of noise can be found.

Code to detect noise already exists in digiKam[14],but it only detects Salt-and-pepper noise using wavelets. Efforts will be made to enable detection of Gaussian and Random noise.

Compression artifacts:

Too much compression can result in visible deterioration of quality of the picture.In this project, we aim to detect errors caused by **JPEG** and **JPEG2000** compression.

- High JPEG compression creates **Blocking** Artifacts.

For its detection, we go through the image, in 8x8 boundaries, if we find that the pixels within that block have more or less the same intensity value, we say that the block is a Blocking Artifact.

- High JPEG2000 compression creates **Ringing** Artifacts.

For its detection, we go through the sharpened image, pixel by pixel, and find edge sequences vertically and horizontally. If edge sequences of sizes between 4 and 64 pixels are found, it is a Ringing Artifact.

The amount of artifacts present is checked with a threshold. If it exceeds the threshold, then there is picture quality loss due to compression(Pages 6-13,[5]).

Considering all these characteristics, the image quality can be determined by applying the formula as described in the paper(Pages 15,16,17 [5]).

Implementation of the algorithm:

The Algorithm will be written as a command-line application using OpenCV, which accepts an image file and displays the result obtained. Any obstacles faced during the implementation of the algorithm will be dealt with by contacting openCV developers and users on IRC(#opencv) as well as researchers by e-mail.Implementing all of the above algorithms would take at maximum 3s for a 256x256 picture on a Pentium IV 3GHZ machine(Page 17,[5]).

Efforts will be made to adapt it to digiKam, by

- using the existing multithreading framework in image parser, and hence allowing multiple images to be processed at the same time.
- enabling the user to selectively choose the algorithms to be used, and hence adjust the speed of the algorithm, depending on his needs.

The existing implementation of Noise Detection in digiKam [14] using wavelets, which detects only Salt-and-pepper noise, will be used and improved to enable detection of Random and Gaussian Noise.

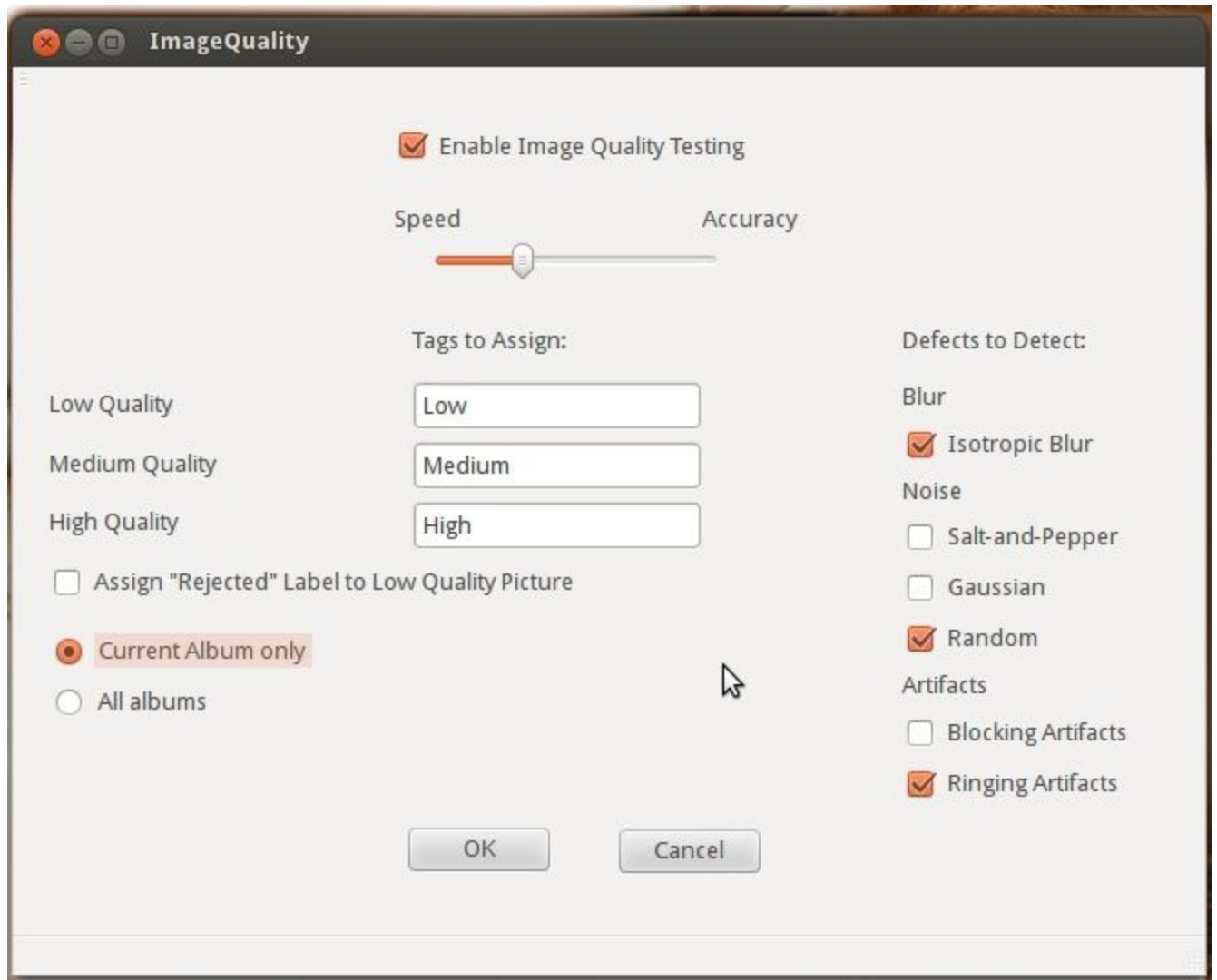
Implementation[Integration into digiKam]:

When the user has enabled this feature in the settings panel,

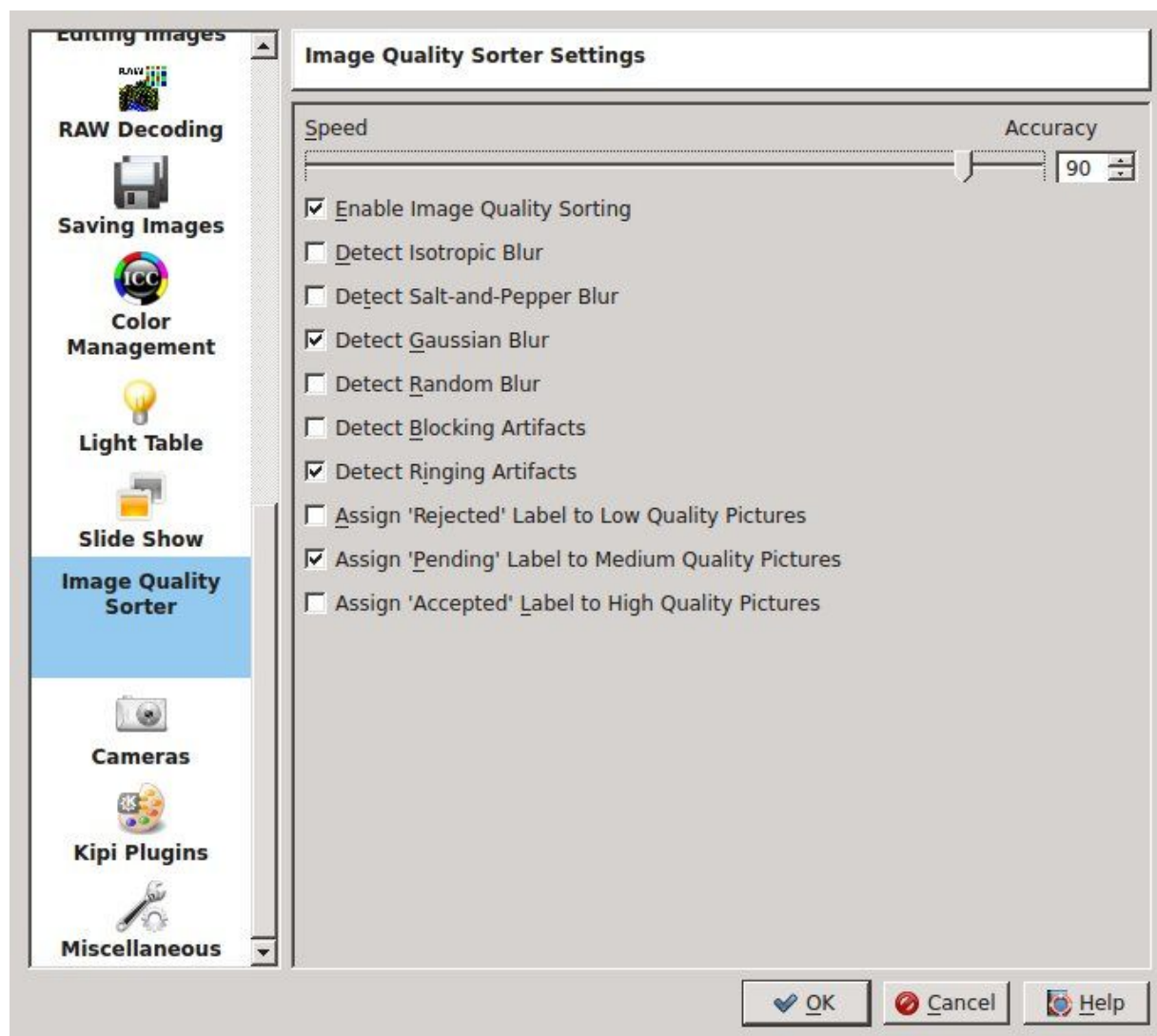
- The Image Scanner interface will be modified to run the algorithm on each new image that passes through it.
- The user can specify whether to pass all the albums through this algorithm or only the current album.
- Each image will be processed and stored with tags specified by the user.
- The digiKam database will store the image quality obtained.
- Options to enable/disable Sharpness, Noise, and Compression defect checking will be provided.
- Low quality pictures are automatically assigned “Rejected” label or not depending on user settings.
- High Image Quality pictures will be displayed initially, then the Medium Image Quality, and then at last, the Low Image Quality Pictures.
- It will be integrated with the Maintenance tool to perform sorting in background. As this project is CPU-intensive, number of threads can be adjusted to ensure the absence of lag while performing other operations.

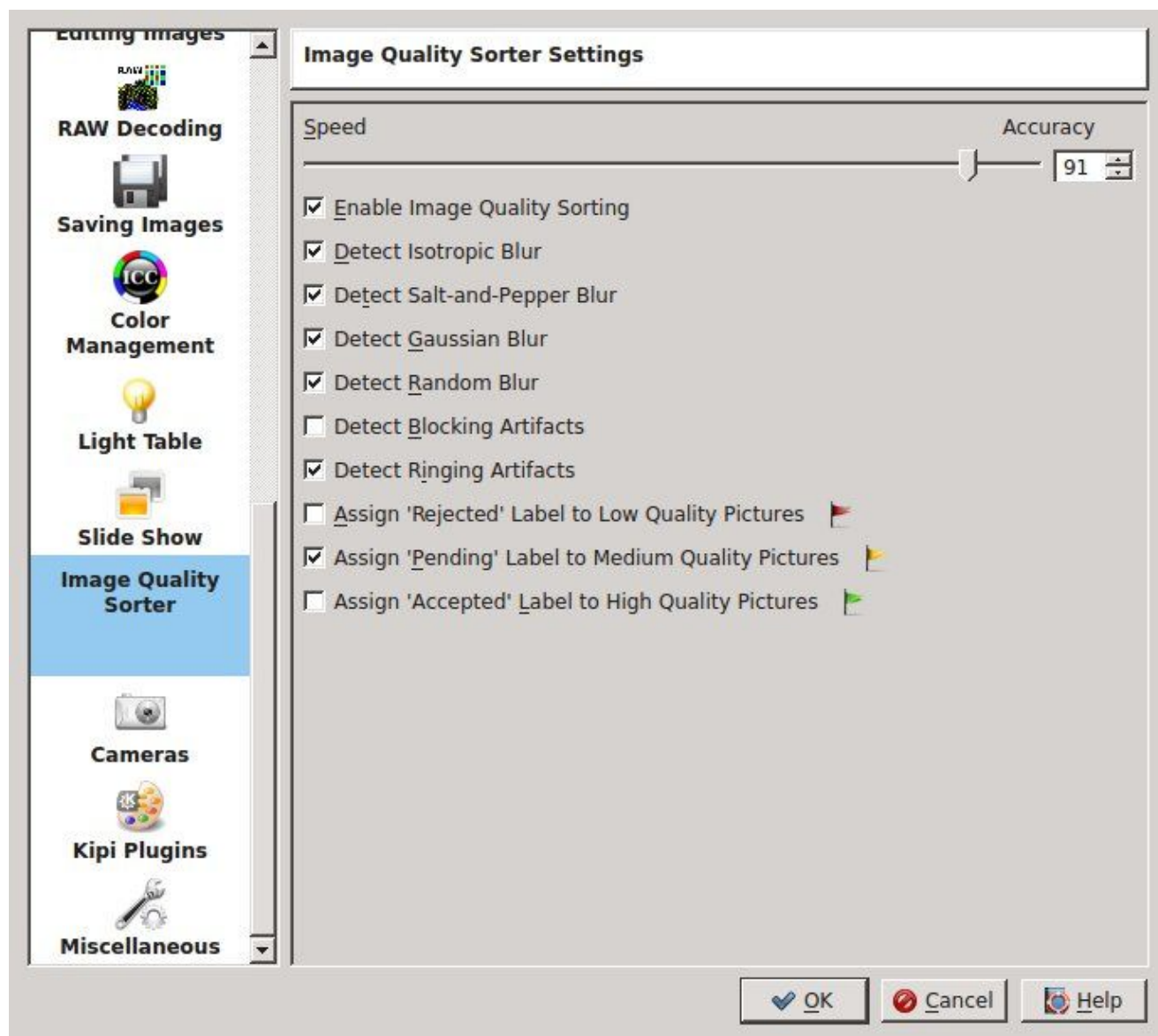
GUI Mockup:

This mockup contains a sample of the proposed settings panel.



Settings Panel






Maintenance Tool



Select Maintenance Operations to Process

▶ ☐  **Scan for new items**

▶ ☐  **Rebuild Thumbnails**


▶ ☐  **Rebuild Finger-prints**

▶ ☐  **Find Duplicates Items**

▶ ☐  **Sync image metadata with Database**


▼ ☒ **Image Quality Sorter**

Image Quality Sorting Management: Skip images already scanned

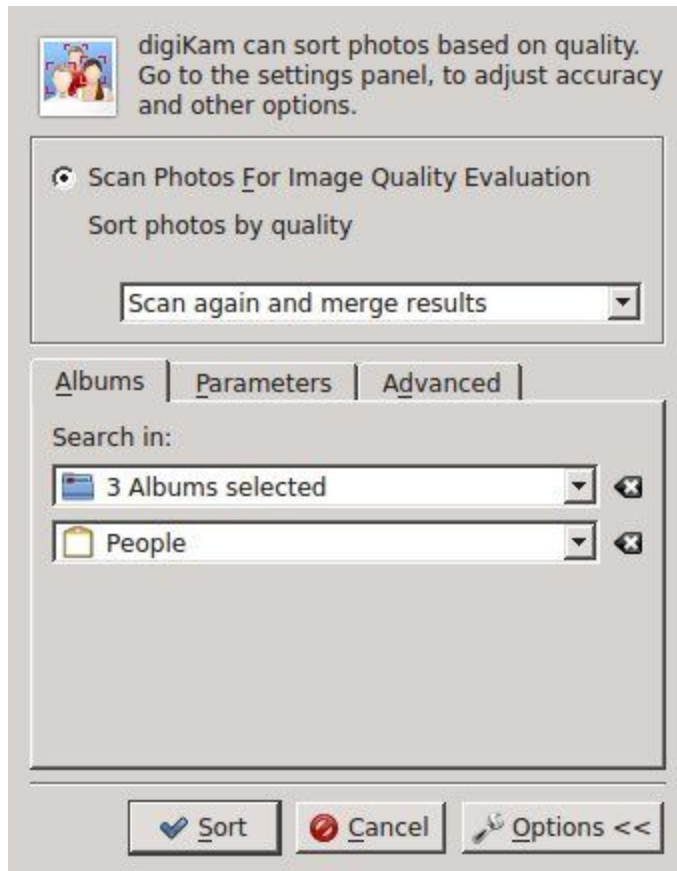
▶ ☐  **Face Detection**

◀ | | ▶

 **OK**

 **Cancel**

 **Help**



Testing:

A collection of pictures taken by various digiKam users, will be collected, and rated on an online survey. The scoring by other users will be hidden, and only visible to the developers of this project. This rating will be compared with the output of this project, and will be calibrated. Tests using QTest will be written to run unit tests and command-line applications will be written to check algorithm and interface.

Timeline:

Upto June 17: (Pre-project research): Go through openCV libraries, write toy programs in it. Go through the Image Parser code in digiKam. Read research papers containing algorithms to detect Image Quality.

June 17-July 4: (Algorithm Implementation) Implementation of the algorithm in digiKam, by writing a stand-alone program using openCV. Conducting survey and getting the human rating of sample pictures. This survey will contain the test cases for the algorithm.

July 4-July 14:(Patching the Scan Interface) The Scan interface will be modified to make the images pass through the Scan Interface. It will also allow users to select a certain number of pictures and pass only those to through the Algorithm.

July 14- July 24:(Assigning Tags) Tags will be added to digiKam Database, and will be fine-tuned, depending on the algorithm's output. I plan to include a general High Quality, Medium Quality, and Low Quality Tag, as well as more specific tags such as Gaussian_Blur_Present depending on the quality of the final Algorithm.

July 24-July 26:(Fine Tuning Algorithm): Based on the output obtained by the Scan Interface, the Algorithm will be improved, and if necessary, multiple algorithms may be used, to provide a balance between speed and accuracy.

July 26- July 29:(Code Review): Reviewing code and progress. Mid-term evaluations.

July 30- August 10:(User options): Working on options to turn off/on specific algorithms will be implemented. Parts of digiKam UI where the newly-created tags would be useful will be patched.

August 10- August 20:(Improving Tags Support) Working on fine-tuning the tags depending on the output obtained during working on the User options. Will collaborate with the Tags Manager developer, if the proposal gets selected, to make this project integrate with it.

August 20- August 30:(Multithreading Support) Code will be modified to support multi-threading framework in digiKam. CPU and Memory usage in a single and multiple threads will be profiled for the sample images.

August 31- September 13:(Testing) The project will be tested by users, released as a special feature, and changes to code will be made based on their feedback. Code will be written to core/tests which performs unit tests and command-line programs will be written to check algorithm and interface.

September 13-September 22:(Pencils down) Cleaning up code, fixing minor bugs, writing documentation.

September 23: Firm 'Pencils down' date.

September 23- October 10:(After GSoC) Efforts will be made to include this feature in the next digiKam stable release.

Extras:

During the duration of GSoC, only if time permits,

- I plan to add algorithms that compromise on accuracy for better speed.
- I plan to add metadata to the image to make it visible other KDE applications
- I plan to add option to modify algorithm from digiKam interface itself.
- Instead of scanning the whole image, I plan to integrate this project with the face detection tool to scan only that area containing the face.[Faster Algorithm]

Do you have other obligations from early June to late September (school, work, vacation, etc.)? Please note that we expect the Summer of Code to be a full-time, 40-hr a week occupation. It is important to be clear and upfront about other commitments that you may have during that time.

I will be working full-time on this project, and will have no classes from early June to late July. My college starts around late July, after which I will be attending college for 4-5 hours. I intend to work late into the evening/night to make up for the lost time.

I will be available approximately 8 hours a day, 6 days a week during the coding period.

About Me:

I am Gowtham Ashok, a sophomore studying Computer Science and Engineering at Madras Institute of Technology, Anna University, India. I come from Chennai, India.

I love programming. Apart from regular college courses and laboratories, I love to write code in my free time. I have written a console-based C++ application[16-bit Turbo C++] in my senior year[12th grade] at school[4], I have also written a C++ application for plagiarism detection[2], a QT C++ Application[3], a quiz program in C and simple Android projects. I am occasionally active in programming contests[12][13].

I've been using KDE for the past 7 years and I like it a lot. I am an open source enthusiast and wish to contribute to this community. I attended a KDE Hackfest held in Shaastra 2013, IIT Madras, have attended various events related to FOSS, and am an active member in the FOSS Society in my college. I have participated in Hackathons.

I use DigiKam for managing my pictures. I have a basic understanding of digiKam's codebase. I regularly follow the digiKam mailing list. I have written a patch for replacing a library.[8][9]. I have also tested patches[10]. My junior jobs link:[11].

Given an opportunity to work on this project, I assure I will put in more than 40 hours per week and that I do not have any other major obligations during the period of the program. I would like to work on

this project even after GSoC and put in efforts to make it available in the next stable digiKam release.

I plan to improve, maintain my code and fix bugs even after the GSoC period is over.

Why am I the right person for the job?

1. I have experience working with C++[2][4] and Qt[3]. I have done database handling with SQLite(Database used in digiKam).I have acquired knowledge of image processing after going through various research papers, and going through digiKam's source code.
2. I am familiar with version control systems, and have used Git regularly.
3. I have experience with openCV [1], and have written a sample algorithm which returns the amount of blur present in an image.
4. I have discussed this project and its technical implementation outline with Gilles Caulier, maintainer of digiKam, Smit Mehta, digiKam developer and a regular digiKam user, Axel Krebs.
5. I have contributed to digiKam by submitting patches. With the help of Gilles Caulier, I have replaced Clapack library with Eigen3 library used in Refocus tool[8][9]. In the process, I went through CMake Documentation and contacted Eigen Developers over IRC. I also tested a patch by Andrew Goodbody, 'Export to jAlbum'[10].

References:

My Projects:

- [1] <https://github.com/gwty/amisharp>
- [2] <https://github.com/gwty/GwtyHash>
- [3] <https://github.com/gwty/Water-Quality-Index>
- [4] <https://github.com/gwty/iDiary>

Research Papers:

- [5] www.cs.uregina.ca/Research/Techreports/2011-02.pdf
- [6] stefan.winklerbros.net/Publications/icip2002.pdf
- [7] 202.194.20.8/proc/ICCT2011/VOL/0080-1569464371.pdf

Bugs:

- [8] https://bugs.kde.org/show_bug.cgi?id=295423
- [9] https://bugs.kde.org/show_bug.cgi?id=251563
- [10] https://bugs.kde.org/show_bug.cgi?id=316719
- [11] <http://community.kde.org/Digikam/GSoC2013>

Myself:

- [12] <http://www.spoj.pl/users/gwty93>
- [13] <http://community.topcoder.com/tc?module=MemberProfile&cr=23011201>

Miscellaneous:

[14]

<https://projects.kde.org/projects/extragear/graphics/digikam/repository/revisions/master/show/libs/dimg/filters/nr>

[15] <http://mail.kde.org/pipermail/digikam-users/2011-February/012238.html>

[16] <http://www.yorba.org/shotwell/help/edit-enhance.html>